NESYPROACT: PROACTIVE NEURAL-SYMBOLIC CONTROL FOR WEB AGENTS

Keyi Xiang[1,*], **Tianyi Tang**[1,*], **Jie-Jing Shao**[1,2], **Yueming Lyu**[1], **Ivor Tsang**[1,3], **Yew-Soon Ong**[1,3], **Haiyan Yin**[1,†]

[1]CFAR and IHPC, Agency for Science, Technology and Research (A*STAR), Singapore
[2]State Key Laboratory of Novel Software Technology, Nanjing University, China
[3]Nanyang Technological University (NTU), Singapore

ABSTRACT

Web-based language agents operate under severe partial observability, where the semantic divergence between internal state assumptions and the ground-truth environment is often unobservable. Consequently, agents frequently maintain high-level reasoning coherence over misgrounded states, causing errors to compound as subsequent decisions are conditioned on silently invalidated premises. We present **NesyProAct**, a proactive neural-symbolic agentic decision framework that transforms conventional ReAct-style web agents from reactive prompting pipelines into execution-aware decision processes. NesyProAct exposes a typed symbolic decision interface over states, actions, transitions, and subgoals, enabling LLM-based agents to automatically compose and reason over execution semantics across decision hierarchies. From this interface, programmatic verification logic is synthesized online to evaluate step-level executability and progress, with verification outcomes directly regulating decision evolution through targeted grounding interventions. As a result, execution feedback is integrated into planning itself, allowing agents to maintain semantic alignment under partial observability rather than propagating unverified assumptions. On WebArena, NesyProAct achieves a **17.8%** overall improvement over strong skill-induction agentic web framework ASI across six domains. NesyProAct also achieves noticeable performance when reasoning with small models GPT-4o-mini, with **5.1%** gain over ASI with Claude-3.5.

1 INTRODUCTION

Web agents operate in partially observable environments governed by latent human-centric interface rules where complex tasks emerge as sequences of interactive transitions Zheng et al. (2025a); Wang et al. (2025a). To succeed, these agents must bridge the abstraction gap between **high-level cognitive reasoning**, defined by the decomposition of goals into subgoals, and **low-level action execution**, characterized by discrete UI interactions Zhou et al. (2024b); Deng et al. (2024). Modern architectures typically delegate this coupling to Large Language Models (LLMs) that must interpret sparse interface feedback to synthesize concrete actions. However, because agents rely on indirect environmental signals, their performance is inherently sensitive to the alignment between latent intent and physical execution.

Most state-of-the-art web agents adopt the **ReAct-style** framework, alternating between reasoning steps and interface actions Yao et al. (2023). While this enables forward-looking planning without explicit environment modeling, it introduces a critical "hallucination" in the decision process: agents often assume that once an action is issued, its intended semantic effect is successfully realized. This **optimistic execution bias** causes the agent's internal state to advance based on assumed outcomes rather than the actual interaction history, leading to catastrophic error propagation when the UI drifts or actions fail Xue et al. (2025); Zhu et al. (2025).

[*]Equal contribution
[†]Corresponding author

These failures reflect a fundamental divergence between the agent's internal decision state and the interaction state induced by execution. When such discrepancies remain undetected, the agent continues to plan and act under invalid assumptions, thereby compounding errors across subsequent decisions. Execution verification offers a principled mechanism to constrain this drift by checking whether intended semantic effects have occurred before the decision state advances. In web environments, grounding conditions are difficult to formalize or observe directly because task-relevant semantics are implicit and only partially revealed through interface feedback. Consequently, many systems rely on language-model-based evaluators to judge execution success post hoc, which effectively reduces verification to another inference problem Xue et al. (2025); Zhu et al. (2025). This *LLM-as-judge* pattern remains reactive and introduces additional uncertainty, cost, and latency.

These observations motivate a paradigm shift in how grounding is modeled within web agent decision making. Rather than treating grounding as a mere perception artifact or a byproduct of improved prompting, execution correctness must be internalized as an **explicit decision variable** that dictates the trajectory of interaction over time. We introduce **NesyProAct**, a neural symbolic control paradigm that restructures the agent's decision space so that execution assumptions become explicit, inspectable, and actionable. NesyProAct represents agent decisions through symbolic interfaces over state conditions, typed actions, expected transitions, and task decomposition constraints, thereby exposing the latent execution semantics that govern web interaction.

From this structured representation, NesyProAct automatically synthesizes **programmatic verification functions** that check both executability and step level semantic progress. Verification outcomes serve as decision shaping signals: detected violations trigger targeted grounding actions that repair execution or acquire missing information before the decision state advances. In this framework, neural components absorb perceptual uncertainty and reference ambiguity from web observations while symbolic structures enforce verification, diagnosis, and control. This architecture replaces brittle heuristics with a **verifiable execution loop** where the agent maintains a provable alignment with the environmental state.

By generalizing over verifiable decision structures and repair operators rather than replaying fragile action scripts, NesyProAct yields substantially more robust behavior under realistic UI drift and partial observability. More broadly, it reframes web interaction as a semantics aware control problem, providing a principled foundation for proactive, verifiable agent behavior in complex environments.

Overall, the contributions of this paper are three-fold:

- We formally define the **Proactive Neural-Symbolic Agentic Decision Process (PNSDP)**, which characterizes agentic web interaction as a decision process where execution semantics, verification, and grounding are explicit components of the transition dynamics under partial observability.
- We introduce **NesyProAct**, an instantiation of PNSDP that automatically synthesizes programmatic verification logic from typed symbolic decision interfaces and induces reusable execution-level skills as verification-governed symbolic programs, enabling structured, hierarchy-aware control over web interactions.
- We demonstrate that NesyProAct substantially improves robustness and generalization on challenging web-agent benchmarks, achieving a 17.8% absolute improvement over strong skill-based baselines on WebArena across six domains under realistic UI drift.

2 Related Works

Web Agent Autonomous web agents have advanced rapidly with the emergence of realistic, high-stakes benchmarks that expose agents to long-horizon, failure-sensitive interactions (Zhou et al., 2024a; Deng et al., 2024; Drouin et al., 2024). Alongside these benchmarks, a diverse set of architectural paradigms has been explored, including ReAct-style interleaved reasoning and action (Yao et al., 2023), reinforcement learning for direct policy optimization (Qi et al., 2025; Yang et al., 2025), and hierarchical planning frameworks that decompose complex objectives into manageable subgoals (Zhang et al., 2025; Erdogan et al., 2025; Zheng et al., 2025b). More recently, trajectory synthesis and data-driven curriculum approaches have been proposed to improve generalization by training agents on diverse interaction traces and counterfactual behaviors (Ou et al., 2024; Xu et al., 2025; Murty et al., 2024a). Together, these efforts have significantly improved benchmark scores and

broadened the scope of tasks that web agents can attempt. However, these gains predominantly operate at the level of reasoning, planning, or data coverage, implicitly assuming that execution faithfully realizes intended action semantics. Recent work further attributes this limitation to the absence of explicit control abstractions, and reformulates interaction as a *Structured Control Graph (SCG)* of typed control commitments with verifiable execution semantics, shifting from action sequence generation to control-centric interaction modeling (Cai et al., 2026).

Despite this progress, the reliability of autonomous web agents remains a critical bottleneck, particularly in settings that require sustained interaction under partial observability. Recent analyses have shown that apparent performance improvements can be brittle, with agents exhibiting abrupt failure modes as task horizon increases or interface dynamics shift (Xue et al., 2025; Zhu et al., 2025). Such failures are not primarily caused by insufficient reasoning capacity, but by a lack of execution grounding: agents often fail to detect subtle but consequential changes in the environment that invalidate previously assumed state transitions. As a result, agents continue to plan and act under misaligned internal states, leading to compounding errors that are difficult to recover from. This limitation has been observed across reasoning-based agents, reinforcement-learning-based systems, and hybrid architectures alike (Qi et al., 2025; Zhang et al., 2025; Yang et al., 2025), suggesting that execution grounding represents a structural challenge rather than an algorithm-specific deficiency.

Skill Induction Building on advances in program synthesis (Ellis et al., 2023; Grand et al., 2024) and embodied skill learning (Wang et al., 2023), recent web agents increasingly induce reusable skills to manage long-horizon interaction. Early systems such as AWM (Wang et al., 2025b) abstract interaction trajectories into reusable workflows, while ASI (Wang et al., 2025a) formalizes skills as executable Python functions that can be invoked compositionally. More recent approaches, including SkillWeaver (Zheng et al., 2025a) and PolySkill (Yu et al., 2025), emphasize cross-site generalization by leveraging guided exploration (Murty et al., 2024b), experience replay (Liu et al., 2025), and tool construction mechanisms (Cai et al., 2024). Collectively, these methods demonstrate that skill induction can substantially reduce planning depth and improve sample efficiency by amortizing interaction patterns across tasks and environments.

However, existing induced skills are typically represented as raw executable code or action scripts, inheriting the same execution fragility as low-level policies. While TroVE (Wang et al., 2024) introduces verification during the induction phase, execution-time correctness remains largely implicit, with no mechanism to detect or repair semantic violations once a skill is deployed. Recent work further highlights that addressing such failures requires structured attribution mechanisms, where execution breakdowns can be localized and repaired at the level of skill structure rather than entire trajectories (Li et al., 2026). As a result, induced skills provide reuse but not reliability: they can fail silently when preconditions are violated or when environmental dynamics deviate from training traces. In contrast, NESYPROACT induces neuro-symbolic skills with explicit preconditions and postconditions, enabling programmatic verification at runtime and targeted repair when execution assumptions break. This shifts skill induction from code reuse toward verifiable decision abstractions, aligning skill execution with the agent's grounding and control loop rather than treating it as an open-loop procedure.NesyProAct induces verifiable neuro-symbolic skills with explicit preconditions/postconditions enabling programmatic verification and targeted repair.

Proactive LLM Agents Long-horizon tasks require belief-state alignment (Shen et al., 2025), which has been addressed through mechanisms such as introspection (Qu et al., 2024), hierarchical memory (Hu et al., 2024), and external memory systems (Chhikara et al., 2025; Fang et al., 2025; Ouyang et al., 2025; Sarch et al., 2024). While these approaches improve temporal coherence, they primarily operate at the level of internal reasoning state and do not directly constrain execution semantics. PAE (Zhou et al., 2024b) introduces LLM-based evaluators for post-hoc judgment, but this design incurs additional computational overhead, stochastic uncertainty, and intervenes only *after* execution errors have already propagated. More broadly, such evaluator-based agents treat verification as a secondary inference problem rather than a control signal that shapes decision flow.

For verification, NesyProAct supplements the stochastic LLM-based evaluators with deterministic symbolic verifiers that operate directly on execution conditions. Verification results are injected into the decision process as control constraints, shaping which actions and skill transitions are admissible at each step. This converts proactive agent behavior from post-hoc reflection into execution-time regulation, preventing belief-state drift from propagating across long-horizon interactions.

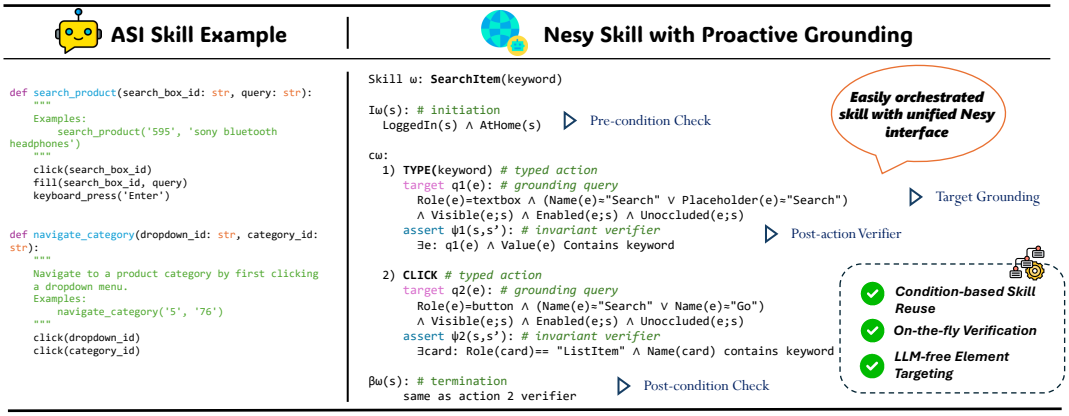

Figure 1: **Comparison between ASI and NeSy-based skill induction.** *Left:* Conventional **Agent Skill Induction (ASI)** treats skills as opaque executable code sequences, implicitly assuming that execution faithfully realizes the intended semantics. As a result, grounding and execution correctness are neither represented nor verified, leading to brittle behavior and poor generalization under environmental variation. *Right:* In contrast, the proposed **NeSy (neuro-symbolic) interface** represents skills as **structured decision objects** with explicit execution semantics, including **typed actions**, **grounding queries**, **pre-condition constraints**, **post-action invariants**, and **termination criteria**. This representation enables an **active grounding-and-verification loop** in which execution outcomes directly shape control decisions, enforcing step-wise semantic correctness and yielding **robust, compositional, and reusable skills**.

3 NesyProAct: Proactive Web Agent with Neural-Symbolic Grounding

In this section, we reformulate **agentic decision making** for web interaction from a *prompt-and-act*, largely reactive pipeline into a **verification-driven**, **proactive grounding-in-the-loop** process, where each step is selected under explicit executability conditions and checked for semantic progress after execution. To this end, we introduce NesyProAct, which is built around four tightly coupled components: (1) a **neural-symbolic decision-making interface** that converts raw UI observations into compact **grounded semantic predicates**, defining the agent's effective decision space; (2) **typed, predicate-grounded action operators** defined directly over this space, with explicit preconditions and expected semantic effects that constrain both planning and execution; (3) a **programmatic verification process** that *automatically synthesizes* executable symbolic verification codes from these predicates and action semantics to validate preconditions and post-action effects, producing structured failure signatures that trigger proactive grounding and minimal repair; and (4) a skill distillation mechanism that packages successful programs into **reusable skills with built-in verification contracts**, so skill execution is governed by the same generated checkers, improving reliability and sample efficiency by replacing most evaluation-time *LLM-as-judge* calls with deterministic checks.

3.1 Proactive Neural-Symbolic Decision Process

We formalize **NesyProAct** as a *Proactive Neural-Symbolic Agentic Decision Process (PNSDP)*, which models web interaction as a partially observable, hierarchical decision-making problem over **typed, verifiable symbolic semantics**. By exposing a structured symbolic interface, this novel proactive grounding framework enables environmental feedback to be systematically integrated into agent's planning trajectory, so execution can be continuously shaped and corrected within the decision process.

Definition 3.1 (Proactive Neural-Symbolic Agentic Decision Process (PNSDP))**.** A PNSDP is a hierarchical decision-making framework defined by the tuple:

$$\mathcal{M}_{\text{PNSDP}} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{G}, \Phi, \mathcal{V}, \Omega \rangle \tag{1}$$

where the operators and spaces are defined as:

- $\mathcal{S}$ and $\mathcal{A}$ denote the latent environment state space and the space of primitive interaction actions, with environment dynamics governed by the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$.
- $\mathcal{O}$ denotes the observation space corresponding to partial UI feedback available after action execution.
- $\mathcal{G}$ denotes the **subgoal space**, where a task specification is decomposed into an ordered set of subgoals that guide high-level planning.
- Φ is a **hierarchical symbolic predicate space** that defines the agent's internal semantic decision interface over states, actions, transitions, and subgoals.
- $\mathcal{V} : \Phi \times \mathcal{A} \times \mathcal{O} \to \mathcal{C}$ is the **programmatic verification operator** that evaluates whether the symbolic effects for an executed action are satisfied.
- Ω denotes the space of grounding and repair operators, corresponding to information-seeking or corrective subgoal actions invoked upon verification outcomes.
- $\mathcal{T}_\Phi : \Phi \times (\mathcal{A} \cup \Omega) \times \mathcal{C} \to \Phi$ defines the **symbolic transition dynamics**, specifying how the agent's state evolves as a function of primitive actions, groundings, and verification outcomes.

Hierarchical Symbolic Predicate Space. The symbolic decision space Φ is structured as a hierarchical predicate library:

$$\Phi = \bigcup_{\ell \in \mathcal{L}} \Phi^{(\ell)}, \qquad \mathcal{L} = \{\mathcal{L}_\mathcal{A}, \mathcal{L}_\mathcal{S}, \mathcal{L}_\mathcal{T}, \mathcal{L}_\mathcal{G}\}, \tag{2}$$

where each level $\Phi^{(\ell)}$ captures semantic constraints at a distinct abstraction level of agentic decision making at the levels of `state` ($\mathcal{L}_\mathcal{S}$), `action` ($\mathcal{L}_\mathcal{A}$), `transition` ($\mathcal{L}_\mathcal{T}$), and `subgoal decomposition` ($\mathcal{L}_\mathcal{G}$). This hierarchical organization exposes execution semantics at multiple abstraction levels, enabling errors to be detected, localized, and corrected proactively by the agent at the appropriate semantic layer.

Each atomic predicate $\phi^{(\ell)} \in \Phi^{(\ell)}$ is defined as a grounded evaluation function:

$$\phi^{(\ell)}(X) = f_{\phi^{(\ell)}}(X), \qquad X \in \mathrm{scope}(\phi^{(\ell)}), \;\; X \sim \mathcal{X}_{\phi^{(\ell)}}, \tag{3}$$

mapping valid inputs to a finite grounding space $\Omega_g^{(\ell)}$. Here, $\mathrm{scope}(\phi^{(\ell)})$ specifies the admissible domain of the predicate (e.g., UI elements, partial observations, action traces, or subplans), while $\mathcal{X}_{\phi^{(\ell)}}$ constrains the input schema through typing and attribute availability. The grounding space $\Omega_g^{(\ell)}$ is typically discrete or boolean-valued and supports programmatic verification.

Verification and Proactive Grounding. NesyProAct synthesizes programmatic verification logic from typed predicates $\phi^{(\ell)} \in \Phi^{(\ell)}$, instantiating execution-relevant constraints into checks whose outcomes directly condition the symbolic transition dynamics. When violations are detected, the grounding operator $\mathcal{G}$ intervenes at the corresponding abstraction level to revise symbolic bindings or repair subgoal structure, making execution semantics explicit control objects under partial observability.

Symbolic Skill Induction. NesyProAct defines skills as reusable execution-level abstractions induced from repeated subgoal execution, represented as *temporally extended symbolic programs over* Φ. Governed by verification-conditioned symbolic transitions, skills remain subject to grounding and verification in the same manner as primitive actions. As a result, skill execution is adaptive to execution context and semantic drift, **rather than being bound to rigid action sequences**, enabling systematic reuse and generalization across tasks and environments.

3.2 Multi-Level Programmatic Symbolic Grounding

NesyProAct introduces a structured symbolic interface that organizes grounding semantics into layered atomic predicates with increasing levels of abstraction. This hierarchy provides the foundational grounding substrate for perception, verification, and proactive control by partitioning the dense observation space into discrete, verifiable components.

We instantiate this grounding structure through four distinct semantic layers:

Decomposition-Level Predicates. Decomposition-level predicates $\phi \in \Phi^{\mathcal{L}_\mathcal{D}}$ determine subtask granularity and dependencies by mapping the *initial observation*, the *target objective*, and the available *typed action space* to a structured decomposition. Concretely, each predicate takes $(o_{\text{init}}, \texttt{obj}, \mathcal{A}_{\text{typed}})$ as input and outputs a *series of subtask intents* together with their dependency relations (e.g., precedence constraints and optional branches), forming an intent sequence that specifies what to achieve next and what must be satisfied before advancing. On the basis of the hierarchical atomic predicate library, the agent compiles logical programs which regulate the agent's belief transition in a structured symbolic space. Appendix A.4 illustrates an example of a subtask with the intent "search for cameras".

$$\phi^{\mathcal{L}_\mathcal{D}} \coloneqq \texttt{obj} \times \mathcal{O}_{\text{init}} \times \mathcal{A}_{\text{typed}} \to \{\texttt{intent}\} \tag{4}$$

Transition-Level Predicates. Transition-level predicates $\phi \in \Phi^{\mathcal{L}_\mathcal{T}}$ take as input a *transition intent*, which is an explicit expectation of the post-transition state after leaving the current page, together with the current observation (starting page). **Rather than validating a given plan, these predicates *synthesize* feasible action sequences that can induce the intended transition**, by enforcing inter-action feasibility and causal consistency under explicit semantic constraints, and outputting candidate typed action sequences with targeting queries based on attributes (e.g., `disabled()=False`, `filled()=True`) instantiated from the focused elements.

$$\phi^{\mathcal{L}_\mathcal{T}} \coloneqq \texttt{intent} \times \mathcal{O} \to \{\mathcal{A}_{\text{typed}} \oplus \texttt{params}\} \tag{5}$$

State-Level Predicates. State-level predicates $\phi \in \Phi^{\mathcal{L}_\mathcal{S}}$ operate on the *current UI observation* (or a scoped part of it, e.g., an accessibility subtree) and perform *action-conditioned element retrieval*. Concretely, each predicate takes as input (i) an observation o_t or $o_t|_{\text{sub}}$, (ii) a typed action, and (iii) the action's parameter-targeting queries expressed as judgements over crafted attributes, and outputs a *series of focused elements* that fit these queries, each paired with its local context required for downstream grounding and verification (e.g., visible labels, placeholders, ancestor traces, and neighborhood cues).

$$\phi^{\mathcal{L}_\mathcal{S}} \coloneqq \mathcal{O} \times \langle \mathcal{A}_{\text{typed}}, \texttt{params} \rangle \to \{\mathcal{E}_{\text{typed}} \oplus \texttt{context}\} \tag{6}$$

Action-Level Predicates. Action-level predicates $\phi \in \Phi^{\mathcal{L}_\mathcal{A}}$ characterize the agent's *action-conditioned belief* over a grounded UI element by reading the element *together with its local context*. Concretely, each predicate takes as input a typed action (e.g., `click`, `type`, `select_option`) and a typed element instance augmented with context (e.g., `MenuItem`, `Button`, `InputBox` plus nearby text, hierarchy, and visual cues), and outputs a crafted attribute belief relevant to that action, such as `disabled()` for `click` feasibility or `filled()` for `type` readiness.

$$\phi^{\mathcal{L}_\mathcal{A}} \coloneqq \langle \mathcal{E}_{\text{typed}}, \texttt{context} \rangle \times \mathcal{A}_{\text{typed}} \to \texttt{attr} \tag{7}$$

3.3 Proactive Grounding-in-the-Loop Decision Making

We embed **grounding** directly into the *decision loop*: the agent first grounds observations into **focused, verifiable signals**, and then makes decisions conditioned on those grounding results. We adopt a **four-layer pipeline**: **decomposition** plans subtasks, **transition** orchestrates skills, **state** performs *skill-conditioned* perception and verification, and **action** executes UI primitives. Crucially, **each upper layer conditions the next**; the chosen subtask or skill determines what is perceived and verified at lower layers, which in turn constrains the feasible actions. Section A.3 presents an algorithmic overview of this hierarchical grounding–control loop.

Interaction Model. We model web interaction as a POMDP with latent states $s_t \in \mathcal{S}$, primitive actions $a_t \in \mathcal{A}_p$, and observations $o_t \in \Omega$:

$$s_{t+1} \sim \mathcal{T}(\cdot \mid s_t, a_t), \qquad o_t \sim \mathcal{O}(\cdot \mid s_t). \tag{8}$$

Since the true state is never directly observed, the agent must rely on structured grounding to extract decision-relevant semantics from partial observations.

Layered Grounding and Decision. Let $\ell \in \{\text{ACTION}, \text{STATE}, \text{TRANSITION}, \text{DECOMPOSITION}\}$ index layers from inner to outer. Each layer ℓ consists of (i) a symbolic predicate library $\mathcal{P}^{(\ell)}$, (ii) a neural grounding module $g_\phi^{(\ell)}$, (iii) a grounding observation space $\Omega_g^{(\ell)}$, (iv) a decision policy $\pi_\theta^{(\ell)}$, and (v) a decision space $\mathcal{D}^{(\ell)}$.

Denote the decision at layer ℓ and time t by $d_t^{(\ell)} \in \mathcal{D}^{(\ell)}$. Each layer operates conditioned on the direct upper-layer decision $d_t^{(\ell+1)}$. Given observation o_t, the grounding module executes a predicate-structured program

$$o_{g,t}^{(\ell)} = \Gamma^{(\ell)}\Big(o_t,\ d_t^{(\ell+1)}; \mathcal{P}^{(\ell)}, g_\phi^{(\ell)}\Big) \in \Omega_g^{(\ell)}, \tag{9}$$

which extracts layer-specific, intent-conditioned semantic representations. The layer policy then produces

$$d_t^{(\ell)} \sim \pi_\theta^{(\ell)}\Big(\cdot \mid o_{g,t}^{(\ell)},\ d_t^{(\ell+1)}\Big). \tag{10}$$

The innermost layer outputs the primitive action:

$$d_t^{(\text{ACTION})} \triangleq a_t \in \mathcal{A}_p. \tag{11}$$

Failure-Aware Replanning and Predicate Expansion. Execution is regulated by structured predicate verification. At each layer ℓ, grounding predicates evaluate whether the current grounding results satisfy the semantic constraints required for the intended decision $d_t^{(\ell)}$. We define a layer-level failure indicator

$$f_t^{(\ell)} = \mathbf{1}\Big[\text{predicate inconsistency between } o_{g,t}^{(\ell)} \text{ and } d_t^{(\ell)}\Big]. \tag{12}$$

If $f_t^{(\ell)} = 1$, the system triggers localized repair at the same layer. Formally, the current decision is revised through a repair operator

$$d_t^{(\ell)} \leftarrow \mathcal{R}^{(\ell)}\big(o_t,\ o_{g,t}^{(\ell)},\ f_t^{(\ell)},\ d_t^{(\ell)}\big), \tag{13}$$

where $\mathcal{R}^{(\ell)}$ denotes an LLM-based repair operator that revises the previously proposed decision conditioned on the structured failure signal.

In some cases, repeated failures may indicate that the existing predicate library $\mathcal{P}^{(\ell)}$ lacks sufficient semantic coverage to express the required grounding constraints. When the repair process determines that the failure cannot be resolved using the current predicates, a new predicate ϕ_{new} may be introduced to capture the missing semantic relation. Formally, the predicate library is extended as

$$\mathcal{P}^{(\ell)} \leftarrow \mathcal{P}^{(\ell)} \cup \{\phi_{\text{new}}\}. \tag{14}$$

The newly introduced predicate becomes available to the grounding program $\Gamma^{(\ell)}$ in subsequent interactions, allowing the agent to progressively improve the expressiveness of its symbolic grounding space. This unified repair mechanism ensures that failures are resolved either by revising the current decision or by expanding the predicate vocabulary when the existing symbolic representation is insufficient.

3.4 Skill Induction/Orchestration with Proactive Grounding

We first define our induced symbolic skill as κ, a reusable neuro-symbolic program compiled from successful trajectories of each subtasks:

$$\kappa \;\coloneqq\; \Big(\ \underbrace{\mathsf{Pre}_\kappa}_{\text{matching condition}}\ ,\ \underbrace{\langle (a_1, q_1), \ldots, (a_T, q_T)\rangle}_{\text{grounded action-query program}},\ \underbrace{\mathsf{Post}_\kappa}_{\text{verifier}}\ \Big). \tag{15}$$

Here, Pre_κ is a symbolic *pre-condition* used for condition-based skill matching, Post_κ is a symbolic *post-condition* for outcome verification, and each program step is a pair (a_t, q_t) consisting of a typed action and its element-targeting query:

$$a_t \in \mathcal{A}_{\text{typed}}, \qquad q_t \in \mathcal{Q}(\Phi^{\mathcal{L}_\mathcal{A}}), \tag{16}$$

where $\mathcal{Q}(\cdot)$ denotes a query language over crafted action-conditioned attributes (e.g., `enabled()`, `filled()`, `occluded()`) with logical operators such as AND, OR, NOT, EXIST, CONTAINS, and IS.

LLM-free Skill Matching and Proactive Reuse. Given a subtask intent τ and current observation o_t, we perform *LLM-free* skill gating by evaluating Pre_κ on the focused element set induced from o_t:

$$\kappa^\star \in \{\kappa \in \mathcal{K} \mid \mathsf{Pre}_\kappa(o_t, \tau) = \texttt{True}\}. \tag{17}$$

where $\mathcal{K}$ is the skill library. Crucially, reuse is not a blind replay: at execution time, each step (a_t, q_t) is *re-grounded* on the current page by retrieving fresh element instances that satisfy q_t, and the action is only committed if its action-level attribute beliefs remain consistent:

$$\mathcal{F}_t \leftarrow \phi_\mathcal{S}(o_t, a_t, q_t), \qquad \texttt{pass}_t \leftarrow \mathsf{Check}\big(\phi_\mathcal{A}(\mathcal{F}_t, a_t)\big). \tag{18}$$

We then execute a_t with instantiated parameters (e.g., `bid`, `text`) derived from $\mathcal{F}_t$ and proceed only if verification passes; otherwise we abort or fall back to planning. After completing the sequence, the post-condition verifier is applied:

$$\textsc{Success}(\kappa, o_{t:T}) \;=\; \mathbf{1}[\mathsf{Post}_\kappa(o_T, \iota) = \texttt{True}]. \tag{19}$$

This design yields *LLM-free skill reuse* through condition-based matching and *proactive grounding* at every step, enabling the same symbolic skill to robustly transfer across page variations without relying on open-loop action replay.

4 Experiments

4.1 Experimental Setup

Benchmark and Evaluation. We evaluate NesyProAct on the WebArena benchmark (Zhou et al., 2024a), which contains 812 test examples spanning five major web activity domains (detailed in Table 1). We also adopted the primitive action space as our starting point without consolidating skills as seen in Appendix A.1.

Following WebArena's standard evaluation protocol, task success is determined using program-based functional correctness evaluators provided by the benchmark. Each test task is associated with a domain-specific deterministic evaluator script that inspects the final environment state after agent execution (e.g., DOM content, page URL, account records, or rendered page elements) and returns a binary success signal indicating whether the task objective has been satisfied. This eliminates reliance on subjective LLM-as-judge scoring and ensures reproducible, execution-grounded evaluation. For tasks involving information retrieval, evaluators verify the presence of correct target entities or values in the final page state; for transactional or navigation tasks, evaluators check that the correct UI state or account-side effect has been reached. We follow the official WebArena and ASI evaluation suite without modification, ensuring full comparability with prior work.

Backbone LM and Agent Architecture. We evaluate NesyProAct with two LM backbones: GPT-4o-mini for cost-effective evaluation and GPT-4o for high-performance assessment. We employ the BrowserGym framework (Drouin et al., 2024) for web interaction and instantiate the primitive action space $\mathcal{A}_p$ with the WebArena default action set (detailed in Appendix A).

Baselines. We compare against three categories of methods (Table 1): reactive single-agent systems (WebRL (Qi et al., 2025), AgentOccam (Yang et al., 2025)), planning-based systems (WebPilot (Zhang et al., 2025)), and skill induction systems (AWM (Wang et al., 2025b), SkillWeaver (Zheng et al., 2025a), ASI (Wang et al., 2025a)). Among these, ASI most directly addresses execution verification, distinguishing our proactive grounding approach from its post-hoc validation strategy.

Methods	LLM Backbone	Domain						Avg.
		Shopping	Admin	Reddit	GitLab	Map	Cross	
Single-Agent Systems								
WebRL (Qi et al., 2025)	Llama-3.1-70B	44.4	54.3	78.9	50.0	40.0	–	49.1
AgentOccam (Yang et al., 2025)	GPT-4-Turbo	40.6	45.6	61.3	37.8	46.8	14.6	43.1
AgentOccam + JUDGE(Yang et al., 2025)	GPT-4-Turbo	43.3	46.2	67.0	38.9	52.3	16.7	45.7
Multi-Agent Systems								
WebPilot (Zhang et al., 2025)	GPT-3.5	25.1	22.0	58.5	30.0	30.3	–	29.1
WebPilot (Zhang et al., 2025)	GPT-4o	36.9	24.7	65.1	39.4	33.9	–	37.2
Web Agentic Systems with Skill Induction								
AWM Wang et al. (2025b)	GPT-4	30.8	29.1	50.9	31.8	43.3	–	35.5
SkillWeaver Zheng et al. (2025a)	GPT-4o	27.2	25.8	50.0	22.2	33.9	–	29.8
ASI Wang et al. (2025a)	Claude-3.5-Sonnet	40.1	44.0	54.7	32.2	43.1	20.8	40.4
Proactive Agentic Systems (Ours)								
NesyProAct (Ours)	GPT-4o-mini	40.6	35.8	72.5	50.5	34.7	22.9	45.5
	GPT-4o	**58.1**	**47.6**	**85.8**	**56.7**	**55.0**	**41.7**	**58.2**
	Claude-3.5-Sonnet	**58.1**	**47.6**	**85.8**	**56.7**	**55.0**	**41.7**	**58.2**

Table 1: Performance comparison across web interaction domains, grouped by agentic framework category. Across all evaluated domains, NESYPROACT consistently outperforms strong skill-induction-based systems, ASI, SkillWeawver and AWM with considerable stability. On `Reddit`, it outperforms ASI by **31.1%**. These results suggest that explicitly modeling grounding and execution verification as part of the decision process yields robustness gains that are not achievable by scaling language models, adding agents, or inducing skills alone.

4.2 BENCHMARK RESULTS

As shown in Tab. 1, NesyProAct achieves the best overall performance among all evaluated methods, establishing a new state of the art on WebArena. With **GPT-4o** as backbone, NesyProAct reaches an average success rate of 58.2%, surpassing the strongest single-agent baseline WebRL (49.1%) and the strongest skill-induction baseline ASI (40.4%) by substantial margins. Notably, NesyProAct also consistently outperforms prior methods across all domain subcategories, demonstrating robust generalization across diverse web interaction environments.

Remarkably, even with the lightweight **GPT-4o-mini** backbone, NesyProAct attains a 45.5% average success rate, matching or exceeding larger-model baselines. This demonstrates that proactive grounding and verification substantially reduce reliance on model scale, enabling small LLMs to achieve strong performance through execution-aware control rather than sheer parametric capacity.

We further report domain-specific breakdowns. In Shopping and Reddit, domains requiring long-horizon navigation and complex operations under dynamic UI states, NesyProAct achieves 58.1% and 85.8%, respectively, surpassing all prior methods. These domains heavily rely on accurate grounding of interface elements. We observe that NesyProAct is particularly effective in such scenarios because predicate-based grounding verification helps prevent common interaction errors, such as clicking incorrect buttons or submitting forms before required fields are completed. By detecting these inconsistencies early during execution, the agent avoids cascading mistakes, which leads to a substantial improvement in overall task success.

n contrast, domains such as Admin remain challenging. These environments often contain hidden interface states, such as menu structures where subcategories are only revealed after interaction. Such latent states are difficult to infer from the current observation, limiting the effectiveness of grounding verification. This suggests that future improvements may require explicit exploration strategies or better modeling of partially observable interfaces.

NesyProAct also achieves state-of-the-art performance on cross-site tasks. We observe that the agent can reuse skills learned from different websites across environments, demonstrating effective cross-site generalization through verification-guided skill reuse. Overall, these results highlight a key advantage of NesyProAct: by decoupling skills from site-specific interfaces and grounding them in verifiable execution semantics, the agent generalizes via decision structure rather than memorization, enabling robust transfer across diverse web environments.

Table 2: **Progressive ablation of NesyProAct on Reddit subset of WebArena using GPT-4o backbone,** where each row removes ("–") one component from the full system and reports the resulting success rate (SR).

NesyProAct Design	**SR(%)**
NesyProAct (Full System)	**85.8**
– Skill Induction	72.6 (-13.2↓)
– State-Level Grounding *(Agent perceives raw partial obs.)*	56.6 (-29.2↓)
– Transition-Level Grounding *(Agent plans w/o transition verifiers)*	49.0 (-36.8↓)

4.3 Ablation Study

As demonstrated in Tab. 2, the full NesyProAct achieves **85.8%** SR. Removing **Skill Induction** reduces SR to **72.6** (**-13.2**), showing skills provide a strong but secondary boost via reusable control. Removing **State-Level Grounding** further drops SR to **56.6** (**-29.2**), indicating that focused element retrieval and context conditioning are critical for reliable targeting. Finally, removing **Transition-Level Grounding** decreases SR to **49.0** (**-36.8**), revealing that verifier-aware action-sequence synthesis is the most influential planning component once elements are grounded. Overall, the largest degradations arise from grounding, implying the performance gains primarily come from proactive grounding rather than open-loop action replay.

The main token cost is from page observation. Applying skill in this situation can significantly deduce the cost, as skill induction does not require additional observation from the environment.

Phases	**LLM calls**	**Token Usage**
Task Decomposition	1	7966.5
Skill Orchestration (*optional if skill matched*)	5.5	28947.6
Execution Verification	6.7	32784.2

The total average token usage without skill reuse is **69,698.3** tokens per task (cold start). **Task Decomposition** is relatively inexpensive, consuming **7,966.5** tokens (**11.4%**) with only **1** call. In contrast, the majority of the cost comes from **Skill Orchestration** (**28,947.6** tokens, **41.5%**) and **Execution Verification** (**32,784.2** tokens, **47.0%**). Since skill reuse is **LLM-free**, relying on condition-based matching and proactive grounding, a matched subtask can skip both orchestration and verification. As a result, each successful skill match removes the dominant cost contributors. Concretely, skill reuse reduces the token cost to **53,930.2** tokens per task.

5 Conclusion

In this work, we studied the problem of reliable decision making for web agents operating in long-horizon, partially observable environments. We introduced **NesyProAct**, which incorporates explicit symbolic grounding and verification into the agent's control loop, allowing execution assumptions to be monitored and corrected during interaction rather than after failure. On the challenging WebArena benchmark, which reflects realistic UI variability and execution uncertainty, NesyProAct achieves more than a 10%+ absolute improvement in success rate over strong reactive, planning-based, and skill-induction baselines. These results suggest that integrating verification-driven control at the decision-process level is a promising direction for building more robust and scalable web agents. In the future, we plan to extend this framework by enabling the agent to autonomously discover and refine its own symbolic verification logic through continuous environmental interaction and self reflection.

ACKNOWLEDGMENTS

This research is supported by the National Research Foundation, Singapore under its AI Singapore Programme (AISG Award No: AISG-NMLP-2024-003), and the National Research Foundation, Singapore and Infocomm Media Development Authority under its Trust Tech Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the National Research Foundation, Singapore, the Agency for Science, Technology and Research, or the Infocomm Media Development Authority.

REFERENCES

Hongtian Cai, Tianyi Ma, Jie-Jing Shao, Tianyi Tang, Ivor Tsang, Yueming Lyu, and Haiyan Yin. Senseact: Structuring gui actions for reliable planning and verification. In *Proceedings of the ICLR 2026 Workshops*, 2026. URL https://openreview.net/forum?id=0DznNQFW4g.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://openreview.net/forum?id=qV83K9d5WB.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Advances in Neural Information Processing Systems (NeurIPS 2023)*, volume 36, 2024.

Alexandre Drouin, Maxime Gasse, Massimo Caccia, Issam H. Laradji, Manuel Del Verme, Tom Marty, Léo Boisvert, Megh Thakkar, Quentin Cappart, David Vazquez, Nicolas Chapados, and Alexandre Lacoste. Workarena: How capable are web agents at solving common knowledge work tasks?, 2024. URL https://arxiv.org/abs/2403.07718.

Kevin Ellis, Lionel Wong, Maxwell Nye, Mathias Sable-Meyer, Luc Cary, Lore Anaya Pozo, Luke Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake–sleep bayesian program learning. *Philosophical Transactions of the Royal Society A*, 381(2251):20220050, 2023.

Lutfi Eren Erdogan, Hiroki Furuta, Sehoon Kim, Nicholas Lee, Suhong Moon, Gopala Anumanchipalli, Kurt Keutzer, and Amir Gholami. Plan-and-act: Improving planning of agents for long-horizon tasks. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=ybA4EcMmUZ.

Runnan Fang, Yuan Liang, Xiaobin Wang, Jialong Wu Manager, Shuofei Qiao, Pengjun Xie, Fei Huang, Huajun Chen, and Ningyu Zhang. Memp: Exploring agent procedural memory, 2025. URL https://arxiv.org/abs/2508.06433. arXiv preprint arXiv:2508.06433.

Gabriel Grand, Lionel Wong, Matthew Bowers, Theo X. Olausson, Muxin Liu, Joshua B. Tenenbaum, and Jacob Andreas. Lilo: Learning interpretable libraries by compressing and documenting code. In *The Twelfth International Conference on Learning Representations (ICLR 2024)*, 2024. URL https://openreview.net/forum?id=TqYbAWKMIe.

Mengkang Hu, Tianxing Chen, Qiguang Chen, Yao Mu, Wenqi Shao, and Ping Luo. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model, 2024. URL https://arxiv.org/abs/2408.09559. arXiv preprint arXiv:2408.09559.

Yuyang Li, Yiran Dou, Jie-Jing Shao, Yueming Lyu, Ivor Tsang, and Haiyan Yin. Skilltracer: Structural failure attribution and refinement of agentic skills in long-horizon web tasks. In *Proceedings of the ICLR 2026 Workshop on Multi-Agent Learning and Its Opportunities in the Era of Generative AI (MALGAI)*, 2026. URL https://openreview.net/forum?id=OiyEjThGeZ.

Yitao Liu, Chenglei Si, Karthik Narasimhan, and Shunyu Yao. Contextual experience replay for self-improvement of language agents, 2025. URL https://arxiv.org/abs/2506.06698. arXiv preprint arXiv:2506.06698.

Shikhar Murty, Dzmitry Bahdanau, and Christopher D. Manning. Nnetscape navigator: Complex demonstrations for web agents without a demonstrator, 2024a. URL https://arxiv.org/abs/2410.02907. arXiv preprint arXiv:2410.02907.

Shikhar Murty, Christopher Manning, Peter Shaw, Mandar Joshi, and Kenton Lee. Bagel: Bootstrapping agents by guiding exploration with language, 2024b. URL https://arxiv.org/abs/2403.08140. arXiv preprint arXiv:2403.08140.

Tianyue Ou, Frank F. Xu, Aman Madaan, Jiarui Liu, Robert Lo, Abishek Sridhar, Sudipta Sengupta, Dan Roth, Graham Neubig, and Shuyan Zhou. Synatra: Turning indirect knowledge into direct demonstrations for digital agents at scale. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems (NeurIPS 2024)*, 2024. URL https://openreview.net/forum?id=KjNEzWRIqn.

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. Reasoningbank: Scaling agent self-evolving with reasoning memory, 2025. URL https://arxiv.org/abs/2509.25140. arXiv preprint arXiv:2509.25140.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, Jie Tang, and Yuxiao Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oVKEAFjEqv.

Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve, 2024. URL https://arxiv.org/abs/2407.18219. arXiv preprint arXiv:2407.18219.

Gabriel Sarch, Lawrence Jang, Michael Tarr, William W. Cohen, Kenneth Marino, and Katerina Fragkiadaki. Vlm agents generate their own memories: Distilling experience into embodied programs of thought. *Advances in Neural Information Processing Systems (NeurIPS 2024)*, 37: 75942–75985, 2024.

Junhong Shen, Hao Bai, Lunjun Zhang, Yifei Zhou, Amrith Setlur, Shengbang Tong, Diego Caples, Nan Jiang, Tong Zhang, Ameet Talwalkar, and Aviral Kumar. Thinking vs. doing: Agents that reason by scaling test-time interaction, 2025. URL http://arxiv.org/abs/2506.07976. arXiv preprint arXiv:2506.07976.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023. URL http://arxiv.org/abs/2305.16291. arXiv preprint arXiv:2305.16291.

Zhiruo Wang, Graham Neubig, and Daniel Fried. Trove: Inducing verifiable and efficient toolboxes for solving programmatic tasks. In *Forty-first International Conference on Machine Learning (ICML 2024)*, 2024. URL https://openreview.net/forum?id=DCNCwaMJjI.

Zora Zhiruo Wang, Apurva Gandhi, Graham Neubig, and Daniel Fried. Inducing programmatic skills for agentic tasks. In *Second Conference on Language Modeling*, 2025a. URL https://openreview.net/forum?id=lsAY6fWsog.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In *Forty-second International Conference on Machine Learning*, 2025b. URL https://openreview.net/forum?id=NTAhi2JEEE.

Yiheng Xu, Dunjie Lu, Zhennan Shen, Junli Wang, Zekun Wang, Yuchen Mao, Caiming Xiong, and Tao Yu. Agenttrek: Agent trajectory synthesis via guiding replay with web tutorials. In *The Thirteenth International Conference on Learning Representations (ICLR 2025)*, 2025. URL https://openreview.net/forum?id=EEgYUccwsV.

Tianci Xue, Weijian Qi, Tianneng Shi, Chan Hee Song, Boyu Gou, Dawn Song, Huan Sun, and Yu Su. An illusion of progress? assessing the current state of web agents, 2025. URL https://arxiv.org/abs/2504.01382. arXiv preprint arXiv:2504.01382.

Ke Yang, Yao Liu, Sapana Chaudhary, Rasool Fakoor, Pratik Chaudhari, George Karypis, and Huzefa Rangwala. Agentoccam: A simple yet strong baseline for LLM-based web agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=oWdzUpOlkX.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://arxiv.org/pdf/2210.03629.

Simon Yu, Gang Li, Weiyan Shi, and Peng Qi. Polyskill: Learning generalizable skills through polymorphic abstraction, 2025. URL https://arxiv.org/abs/2510.15863.

Yao Zhang, Zijian Ma, Yunpu Ma, Zhen Han, Yu Wu, and Volker Tresp. Webpilot: A versatile and autonomous multi-agent system for web task execution with strategic exploration. In *AAAI*, pp. 23378–23386, 2025. URL https://doi.org/10.1609/aaai.v39i22.34505.

Boyuan Zheng, Michael Y. Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, and Yu Su. Skillweaver: Web agents can self-improve by discovering and honing skills, 2025a. URL https://arxiv.org/abs/2504.07079. arXiv preprint arXiv:2504.07079.

Chengqi Zheng, Jianda Chen, Yueming Lyu, Wen Zheng Terence Ng, Haopeng Zhang, Yew-Soon Ong, Ivor Tsang, and Haiyan Yin. Mermaidflow: Redefining agentic workflow generation via safety-constrained evolutionary programming, 2025b. URL https://arxiv.org/abs/2505.22967.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2024a. URL https://arxiv.org/abs/2307.13854.

Yifei Zhou, Qianlan Yang, Kaixiang Lin, Min Bai, Xiong Zhou, Yu-Xiong Wang, Sergey Levine, and Erran Li. Proposer-agent-evaluator(pae): Autonomous skill discovery for foundation model internet agents, 2024b. URL http://arxiv.org/abs/2412.13194. arXiv preprint arXiv:2412.13194.

He Zhu, Tianrui Qin, King Zhu, Heyuan Huang, Yeyi Guan, Jinxiang Xia, Yi Yao, Hanhao Li, Ningning Wang, Pai Liu, Tianhao Peng, Xin Gui, Xiaowan Li, Yuhui Liu, Yuchen Eleanor Jiang, Jun Wang, Changwang Zhang, Xiangru Tang, Ge Zhang, Jian Yang, Minghao Liu, Xitong Gao, Jiaheng Liu, and Wangchunshu Zhou. Oagents: An empirical study of building effective agents, 2025. URL https://arxiv.org/abs/2506.15741. arXiv preprint arXiv:2506.15741.

APPENDIX

A IMPLEMENTATION DETAILS

A.1 PRIMITIVE ACTION SPACE

Our agent operates using a standard set of primitive actions from the WebArena benchmark, which enables interaction with web interfaces through the BrowserGym framework. Table 3 enumerates the complete action space available to the agent. These actions are organized into four functional categories: basic element interactions, page navigation, tab management, and user communication.

Table 3: Primitive action space $\mathcal{A}_p$ used throughout our experiments.

ACTION	FUNCTIONALITY
Basic Interactions	
NOOP(WAIT_MS)	IDLE FOR SPECIFIED DURATION.
CLICK(ELEM)	ACTIVATE AN ELEMENT.
HOVER(ELEM)	MOVE CURSOR OVER AN ELEMENT.
FILL(ELEM, VALUE)	INPUT TEXT INTO AN ELEMENT.
KEYBOARD_PRESS(KEY_COMB)	EXECUTE A KEY COMBINATION.
SCROLL(X, Y)	SCROLL IN HORIZONTAL OR VERTICAL DIRECTION.
SELECT_OPTION(ELEM, OPTIONS)	CHOOSE FROM AVAILABLE OPTIONS.
Page Navigation	
GOTO(URL)	NAVIGATE TO SPECIFIED URL.
GO_BACK()	RETURN TO PREVIOUS PAGE.
GO_FORWARD()	ADVANCE TO NEXT PAGE.
Tab Management	
NEW_TAB()	CREATE A NEW BROWSER TAB.
TAB_CLOSE()	CLOSE ACTIVE TAB.
TAB_FOCUS(INDEX)	SWITCH TO SPECIFIED TAB.
User Communication	
SEND_MSG_TO_USER(TEXT)	DELIVER MESSAGE TO USER.
REPORT_INFEASIBLE(REASON)	INDICATE TASK CANNOT BE COMPLETED.

A.2 TABLE OF DEFINITIONS FOR PROACTIVE NEURAL-SYMBOLIC AGENTIC DECISION SPACE (PNSDP)

Symbol	Formal Definition and Role
$\mathcal{S}, \mathcal{A}$	Latent environment state space and primitive interaction action space, with dynamics governed by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$.
$\mathcal{O}$	Observation space corresponding to partial UI feedback available to the agent after action execution.
$\mathcal{G}$	Subgoal space where task specifications are decomposed into an ordered set of subgoals to guide high level planning.
Φ	Hierarchical symbolic predicate space defining the internal semantic decision interface over states, actions, transitions, and subgoals.
$\mathcal{V}$	Programmatic verification operator $\mathcal{V} : \Phi \times \mathcal{A} \times \mathcal{O} \to \mathcal{C}$ that evaluates whether symbolic effects for an executed action are satisfied.
Ω	Space of grounding and repair operators, corresponding to information seeking or corrective subgoal actions invoked upon verification outcomes.
$\mathcal{T}_\Phi$	Symbolic transition dynamics $\mathcal{T}_\Phi : \Phi \times (\mathcal{A} \cup \Omega) \times \mathcal{C} \to \Phi$ specifying the evolution of the agent's internal state.

Table 4: Formal components of the Proactive Neural-Symbolic Agentic Decision Process (PNSDP) derived from Definition 3.1.

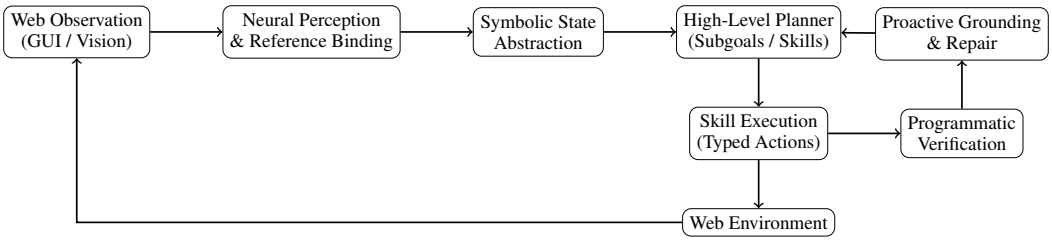

Figure 2: Overview of the hierarchical planning pipeline with neural-symbolic proactive grounding in NESYPROACT. High-level planning operates over symbolic abstractions, while low-level execution is continuously verified and repaired through an explicit grounding loop.

A.3 OVERVIEW OF HIERARCHICAL PLANNING PIPELINE WITH NEURAL SYMBOLIC PROACTIVE GROUNDING

Figure 2 illustrates the hierarchical planning pipeline underlying NESYPROACT, highlighting how neural perception, symbolic decision making, and proactive grounding are integrated into a single closed decision loop. A key property of this design is the explicit separation between perceptual uncertainty handling and decision-level verification: neural components are used solely to bind observations to symbolic references, while symbolic structures govern state abstraction, action typing, transition verification, and repair. This separation ensures that execution correctness is enforced at the decision level rather than inferred post hoc from language-model judgments. Another important property is temporal stability: verification outcomes feed back directly into high-level planning, preventing unverified execution assumptions from propagating across steps and enabling controlled recovery before errors compound. Looking ahead, this pipeline opens several directions for future work, including learning adaptive grounding strategies, extending symbolic interfaces to richer temporal or probabilistic constraints, and scaling proactive verification to multi-agent or long-running web workflows.

A.4 SYMBOLIC GROUNDING PROGRAM DEMONSTRATION

Program A.1 (Autonomous Grounding Program (compiled example)). **Subtask Intent:** `Search for ``camera'' and open the first result.`

Transition-Level Program: `CompileTransitionIntent`$(\iota, \mathcal{F}) \mapsto (\mathbf{a}_{1:T}, \mathbf{q}_{1:T})$

Step 1	**Type query into search box**
Action	`type`(`bid` $= \$b_1$, `text` $=$ "camera")
Query	`IS(InputBox) AND enabled()=True AND occluded()=False AND (CONTAINS(placeholder,"search") OR CONTAINS(label,"search")) AND filled()=False`
Step 2	**Trigger search**
Action	`click`(`bid` $= \$b_2$)
Query	`IS(Button) AND enabled()=True AND occluded()=False AND (CONTAINS(text,"search") OR CONTAINS(aria_label,"search"))`
Guard	`NOT EXIST( IS(Button) AND (CONTAINS(text,"search") OR CONTAINS(aria_label,"search")) )` $\Rightarrow$ `use fallback action: press(key=ENTER)`
Step 3	**Open first result**
Action	`click`(`bid` $= \$b_3$)
Query	`IS(Link) AND enabled()=True AND occluded()=False AND (CONTAINS(text,"camera") OR CONTAINS(desc,"camera")) AND rank=Top1`
Output	$(\mathbf{a}_{1:3}, \mathbf{q}_{1:3}) = ([\texttt{type}(\$b_1, \text{"}camera\text{"}), \texttt{click}(\$b_2), \texttt{click}(\$b_3)], [q_1, q_2, q_3])$

Algorithm 1 Algorithm workflow of NESYPROACT with four-level grounding

Require: Task objective $\mathcal{G}$, initial observation o_0, augmented typed action space $\mathcal{A}_{\text{typed}} \cup \mathcal{A}_{\text{skill}}$, atomic predicate library Φ
Ensure: Completed task (or best-effort trajectory), updated symbolic skill set $\mathcal{A}_{\text{skill}}$

/* Level-4 Grounding: Decomposition-level (subtask planning) */
$\{\iota_k\}_{k=1}^{K}, \mathcal{E} \leftarrow \phi_{\mathcal{D}}(o_0, \mathcal{G}, \mathcal{A}_{\text{typed}} \cup \mathcal{A}_{\text{skill}})$ ▷ output intents & dependencies
Initialize focused element set $\mathcal{F}_0 \leftarrow \varnothing$
for $k \leftarrow 1$ **to** K **do**
  /* Subtask intent */ ι_k
  /* NeSy skill gating: condition-based matching */
  $s^\star \leftarrow \text{MATCHSKILL}(\iota_k, o_t, \mathcal{A}_{\text{skill}}, \Phi)$
  if $s^\star \neq \varnothing$ **then**
    /* LLM-free skill reuse: purely symbolic control */
    $(o_t, \texttt{ok}) \leftarrow \text{EXECUTESKILL}(s^\star, o_t, \Phi)$
    if `ok` **then**
      continue
    end if
  end if
  /* Level-3 Grounding: Transition-level (skill planning / orchestration) */
  $(\mathbf{a}_{1:T}, \mathbf{q}_{1:T}) \leftarrow \phi_{\mathcal{T}}(\iota_k, \mathcal{F}_t)$ ▷ output typed action sequence + parameter queries
  for $t' \leftarrow 1$ **to** T **do**
    /* Level-2 Grounding: State-level (focused element retrieval) */
    $\mathcal{F}_{t'} \leftarrow \phi_{\mathcal{S}}(o_{t'}, a_{t'}, q_{t'})$ ▷ output elements + context fitting queries
    /* Level-1 Grounding: Action-level (attribute belief / verification) */
    $\texttt{attr}_{t'} \leftarrow \phi_{\mathcal{A}}(\mathcal{F}_{t'}, a_{t'})$ ▷ e.g., `disabled()`, `filled()`
    $\hat{o}_{t'+1} \leftarrow \text{ACT}(a_{t'};\, \mathcal{F}_{t'})$
    $\texttt{pass} \leftarrow \text{VERIFY}(a_{t'}, \texttt{attr}_{t'}, \hat{o}_{t'+1})$
    if not `pass` **then**
      break
    end if
    $o_{t'+1} \leftarrow \hat{o}_{t'+1}$
  end for
  if $\text{SUBTASKSUCCESS}(\iota_k, o_t, \Phi)$ **then**
    /* Skill induction: store or update symbolic skill */
    $\mathcal{A}_{\text{skill}} \leftarrow \text{STOREORUPDATESKILL}(\iota_k, \tau_k, \Phi, \mathcal{A}_{\text{skill}})$
  end if
end for

A.5 PROMPT DETAILS

Prompt 1: Skill Reuse

You are a skill reuse expert who determines whether a skill can be reused to complete the current subtask.
You are given: - The overall task goal. - The query of the current subtask. - All available skills in the current situation, their descriptions, and parameters(If any).
You will return: - Whether a skill can be reused, the skill name and parameter if any. **Critical**: Return only the EXACT skill name if it can be reused and null if no skill can be reused.
Instructions: - If a skill can be reused, the skill name and description should match EXACTLY with the intention of the subgoal. - If a skill has parameters, the skill name should include the parameter based on the given query and task goal. - If a skill dont require parameters, set the parameter to null. - **Critical**: Try your best to use the skill if possible. - If no skill can be reused, set the name to null and parameter to null.
Example: Subtask goal: Search for laptops Skills available: search_for_items(parameter required: True), find_most_expensive_book(parameter required: False) Response: "name":"search_for_items", "parameter":"laptop"
Subtask goal: Navigate to review page Skills available: Navigate_to_review_page(parameter required: False), Navigate_to_user_account(parameter required: False) Response: "name":"Navigate_to_review_page", "parameter":null

Output format strictly in JSON format,do not output any other context: ”name”:”skill_name”, ”parameter”:”parameter” ”””

Prompt 2: Planner

You are a task planner specialized in web task decomposition. Your task is to decompose a complex web task into multiple subtasks with clear dependency and execution order relationships.
You are given: - The overall task goal - The current URL of the web page.
You will return: - A list of subtasks in JSON format
Instructions: - Each subtask should have a one-sentence description, it should be concise and clear(one verb + what to do). For example, ”Search for wireless earphone”, ”Navigate to the review page”, etc. - The subtask should be concise and clear, describing a specific web subgoal. For example, ”Sort products by price”, instead of ”Sort products”. - Login in is NOT required, the web page is automatically logged in.
For each subtask, you MUST: Provide precondition_predicates: These predicates verify that necessary conditions are met before executing this subtask by checking if certain information exists: - Previous action sequence verification (check if certain action types were executed) - URL verification (check if URL contains certain keywords) - Current page accessibility tree verification (check if axtree contains certain keywords)
Precondition predicates format (all fields check for existence, not complex logic): - ”previous_actions”: List of action type strings that MUST appear in previous action sequences - Example: [”fill”, ”click”] means ”fill” and ”click” action types must have been executed in previous subgoals - This verifies the execution flow: e.g., before ”sort”, we need to have ”search” (which typically involves ”fill” action) - Common action types: ”click”, ”fill”, ”hover”, ”go_back”, ”go_forward”, ”noop” - Check: All listed action types must exist in the previous action sequences - ”url”: List of keywords that MUST appear in the current URL - Example: [”earphone”, ”search”] means both ”earphone” and ”search” must be present in the URL - This verifies that we are on the correct page after previous actions - Use this to ensure we are on a search results page, product page, etc. - Check: All listed keywords must be found in the current URL - ”axtree”: List of keywords that MUST appear in the current page accessibility tree - Example: [”earphone”, ”price”] means both ”earphone” and ”price” must be present in the current page axtree - This verifies that expected content is visible on the current page - Check: All listed keywords must be found in the current page axtree (as text content) - Key words must be SINGLE word, not phrases. - For example, if the key word is galaxy-s6-screen-protector, dismantal it into [”galaxy”, ”s6”, ”screen”, ”protector”]
IMPORTANT: - All precondition checks are simple existence checks - if the field is specified, all listed items must be found - Use ”previous_actions” to verify execution flow (e.g., [”fill”] before sort means search must have been done) - Use ”url” to verify page state (e.g., [”search”] to ensure we’re on search results page) - Use ”axtree” to verify page content (e.g., [”earphone”] to ensure search results are displayed)
Please return the subtasks in JSON format. Example: ”subtasks”: [”id”: 1, ”description”: ”Search for wireless earphone”, ”precondition_predicates”: null , ”id”: 2, ”description”: ”Sort products by price”, ”precondition_predicates”: ”previous_actions”: [”fill”], ”url”: [”earphone”, ”search”], ”axtree”: [”earphone”]]

Prompt 3: Subtask Evaluator

You are a web task judge that can judge whether a subgoal has been reached by inspecting the resulting web page.
You are given: - The executed subgoal - The resulting web page represented as an accessibility tree with every element whose ID, role, and value are specified. - The excuted action history of the current subtask.
Instructions: - First focus on some key elements that may be indicative of the subgoal being reached. - Output a rationale for the thinking process and your judgement out of Failed, Success in ¡¿.
Critical: If the subgoal is modification of the current page, you should judge the success based on the action history. If the action history already completed the subgoal

Below are the judgement candidates,only one of them should be selected: ¡success¿ ¡failed¿ ¡incomplete¿
Example: Executed subgoal: Search for laptops
Rationale: The subgoal is to search for laptops. I need to check if the search results are now visible and accessible on the page. Looking at the accessibility tree, I can see some elements with role ”listitem” and values like ”Dell XPS 13”, ”MacBook Pro”, ”HP Spectre x360”, etc. This indicates the search results have been successfully displayed. ¡success¿